

Chapter 3: Strings, Vectors and Arrays

3.1 Namespace using Declaration

```
#include <iostream>

// using declarations for names from the standard library
using std::cin;
using std::cout;
using std::endl;
int main()
{
    cout << "Enter two numbers:" << endl;

    int v1, v2;
    cin >> v1 >> v2;

    cout << "The sum of " << v1
         << " and " << v2
         << " is " << v1 + v2 << endl;

    return 0;
}
```

Or using entire std namespace

```
#include <iostream>

// using declarations for the entire standard library
using namespace std;

int main()
{
    cout << "Enter two numbers:" << endl;

    int v1, v2;
    cin >> v1 >> v2;

    cout << "The sum of " << v1
         << " and " << v2
         << " is " << v1 + v2 << endl;

    return 0;
}
```

(see ppt)

3.2 Library string Type

string I/OStringIO.cpp

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string s;
    cin >> s;
    cout << s << endl;
    return 0;
}
```

Q: what are the outputs if we enter Hello World! from inputs?

A:

Now another example:

StringGetLine.cpp

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string line;
    getline(cin, line);
    cout << line << endl;
    return 0;
}
```

Q: what are the outputs if we enter Hello World! from inputs?

A:

Quick ChecksStringIOEx1.cpp

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string s;
    while (cin >> s)
        cout << s << endl;
    return 0;
}
```

```
| }
```

StringIOEx2.cpp

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string s;
    while (getline(cin, s))
        cout << s << endl;
    return 0;
}
```

Q1: What does the above code fragment intend to achieve?

A1:

Q2: How to stop the program?

A2:

Table 3.2 string Operation

<code>os << s</code>	Writes <code>s</code> onto output stream <code>os</code> . Returns <code>os</code> .
<code>is >> s</code>	Reads whitespace-separated string from <code>is</code> into <code>s</code> . Returns <code>is</code> .
<code>getline(is, s)</code>	Reads a line of input from <code>is</code> into <code>s</code> . Returns <code>is</code> .
<code>s.empty()</code>	Returns <code>true</code> if <code>s</code> is empty; otherwise returns <code>false</code> .
<code>s.size()</code>	Returns the number of characters in <code>s</code> .
<code>s[n]</code>	Returns a reference to the char at position <code>n</code> in <code>s</code> ; positions start at 0.
<code>s1 + s2</code>	Returns a string that is the concatenation of <code>s1</code> and <code>s2</code> .
<code>s1 = s2</code>	Replaces characters in <code>s1</code> with a copy of <code>s2</code> .
<code>s1 == s2</code>	The strings <code>s1</code> and <code>s2</code> are equal if they contain the same characters.
<code>s1 != s2</code>	Equality is case-sensitive.
<code><, <=, >, >=</code>	Comparisons are case-sensitive and use dictionary ordering.

string size Operation:

It might be logical to expect that `s.size()` returns an `int` or an `unsigned`. Instead,

`s.size()` returns a `string::size_type` value. The reason is that the `string` class—and most other library types—defines several **companion types**. These companion types make it possible to use the library types in a machine independent manner. The type `size_type` is one of these companion types.

To use the `size_type` defined by `string`, we use the scope operator to say that the name `size_type` is defined in the `string` class. Although we don't know the precise type of `string::size_type`, **we do know that it is an unsigned type big enough to hold the size of any string.**

It can be tedious to type `string::size_type`. Under the new standard, we can ask the compiler to provide the appropriate type by using `auto`

```
string s = "I am a C++ string";  
string::size_type len1 = s.size(); //C++98  
auto len2 = s.size(); // len has type string::size_type, C++11
```

Exercise 3.1 In-class Coding Exercise

Ex31.cpp

Write a program to read strings from the standard input, add their size and merge what is read into one large string. Use ! to terminate the input. Print the input strings, their size, merged string and its size. A sample run looks like:

```
Enter a few strings and terminate with !: I love C++ programming !  
I size is: 1  
love size is: 4  
C++ size is: 3  
programming size is: 11  
The merged string is: IloveC++programming and its size is: 19
```

Answer:

(see ppt)

3.2.3 Dealing with Characters in a string

Table 3.3 ctype Functions

<code>isalnum(c)</code>	true if <code>c</code> is a letter or a digit.
<code>isalpha(c)</code>	true if <code>c</code> is a letter.
<code>iscntrl(c)</code>	true if <code>c</code> is a control character.
<code>isdigit(c)</code>	true if <code>c</code> is a digit.
<code>isgraph(c)</code>	true if <code>c</code> is not a space but is printable.
<code>islower(c)</code>	true if <code>c</code> is a lowercase letter.
<code>isprint(c)</code>	true if <code>c</code> is a printable character (i.e., a space or a character that has a visible representation).
<code>ispunct(c)</code>	true if <code>c</code> is a punctuation character (i.e., a character that is not a control character, a digit, a letter, or a printable whitespace).
<code>isspace(c)</code>	true if <code>c</code> is whitespace (i.e., a space, tab, vertical tab, return, newline, or formfeed).
<code>isupper(c)</code>	true if <code>c</code> is an uppercase letter.
<code>isxdigit(c)</code>	true if <code>c</code> is a hexadecimal digit.
<code>tolower(c)</code>	If <code>c</code> is an uppercase letter, returns its lowercase equivalent; otherwise returns <code>c</code> unchanged.
<code>toupper(c)</code>	If <code>c</code> is a lowercase letter, returns its uppercase equivalent; otherwise returns <code>c</code> unchanged.



If we want to do something to every character in a string, by far the best approach is to use a statement introduced by the new standard: the **range for** statement. This statement iterates through the elements in a given sequence and performs some operation on each value in that sequence. The syntactic form is

```
for (declaration : expression)
    statement
```

(see ppt)

Exercise 3.2 In-class Coding Exercise

Ex32.cpp

Write a program to read a line of strings from the standard input and count the number of punctuations in the line. A sample run looks like:

```
Enter a line of strings: Hello, my name is Doris!!!  
The number of punctuations is: 4
```

Answer:

(see ppt)

3.3.1 Defining and Initializing vectors

Table 3.4 The ways to Initialize a vector

<code>vector<T> v1</code>	vector that holds objects of type T. Default initialization; v1 is empty.
<code>vector<T> v2 (v1)</code>	v2 has a copy of each element in v1.
<code>vector<T> v2 = v1</code>	Equivalent to <code>v2 (v1)</code> , v2 is a copy of the elements in v1.
<code>vector<T> v3 (n, val)</code>	v3 has n elements with value val.
<code>vector<T> v4 (n)</code>	v4 has n copies of a value-initialized object.
<code>vector<T> v5 {a,b,c ... }</code>	v5 has as many elements as there are initializers; elements are initialized by corresponding initializers.
<code>vector<T> v5 = {a,b,c ... }</code>	Equivalent to <code>v5 {a,b,c ... }</code> .

Table 3.5 vector Operation

<code>v.empty()</code>	Returns true if v is empty; otherwise returns false.
<code>v.size()</code>	Returns the number of elements in v.
<code>v.push_back(t)</code>	Adds an element with value t to end of v.
<code>v[n]</code>	Returns a reference to the element at position n in v.
<code>v1 = v2</code>	Replaces the elements in v1 with a copy of the elements in v2.
<code>v1 = {a,b,c ... }</code>	Replaces the elements in v1 with a copy of the elements in the comma-separated list.
<code>v1 == v2</code>	v1 and v2 are equal if they have the same number of elements and each element in v1 is equal to the corresponding element in v2.
<code>v1 != v2</code>	
<code><, <=, >, >=</code>	Have their normal meanings using dictionary ordering.

Using **push_back** member function

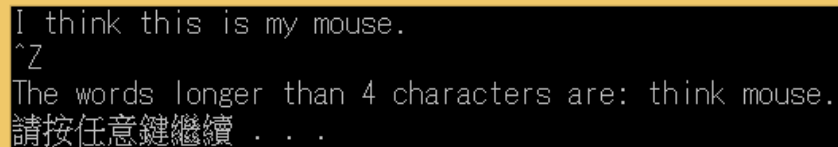
To store a value in a vector that does not have a starting size or that is already full, you should use the `push_back` member function. This function accepts a value as an argument and store it in a new element placed at the end of vector. It “**pushes**” the value at the “**back**” of the vector. For example,

```
vector<int> x;
x.push_back(12);
```

Q: what happens?

A:

With introduction of `string` and `vector`, we can easily store word from standard input into the `vector` container and process these words upon request. For example, we can ask users to input a few words, store them in a `vector` and parse and print those words that are longer than 4 characters.

A terminal window with a black background and yellow text. The first line shows the user input "I think this is my mouse." followed by a carriage return (^Z). The second line shows the program output "The words longer than 4 characters are: think mouse." followed by a prompt "請按任意鍵繼續 . . ." (Press any key to continue . . .).

```
I think this is my mouse.  
^Z  
The words longer than 4 characters are: think mouse.  
請按任意鍵繼續 . . .
```

VectorStringEx.cpp

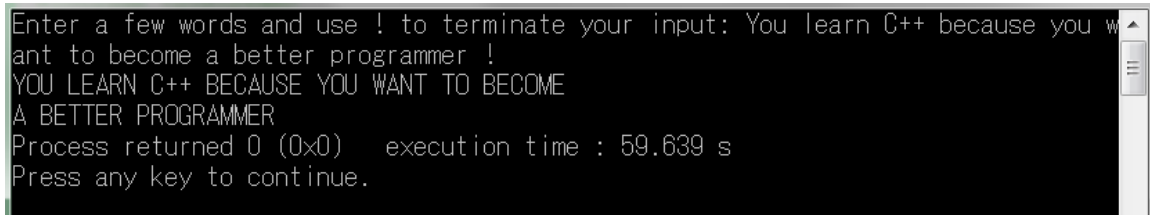
```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int main()
{
    string word;
    vector<string> text;
    while (cin >> word)
        text.push_back(word);
    for (auto s : text)
        if (s.size() > 4) cout << s << endl;
    return 0;
}
```

Exercise 3.3 In-class Coding Exercise

Ex33.cpp

Read a sequence of words from `cin` and store the values a `vector`. After you've read all the words, process the vector and change each word to uppercase. Finally, print the transformed elements, eight words to a line. A sample run looks like:



```
Enter a few words and use ! to terminate your input: You learn C++ because you want to become a better programmer !
YOU LEARN C++ BECAUSE YOU WANT TO BECOME
A BETTER PROGRAMMER
Process returned 0 (0x0) execution time : 59.639 s
Press any key to continue.
```

Answer: